Small Files Aggregation Operations Guide.

Table of Contents

1 Introduction

This project is primarily driven by the need to aggregate small files into bigger packages for more efficient use of tape drives.

Tape drive transfer rates depend on the size of the files. If the size of a file is relatively small then overall average data throughput rate is less than for the relatively big file size. There are several reasons for this: seeking to the files position, starts/ stops etc. The optimal size for a file on tape depends on the tape technology and currently is about 1GB with tendency to grow[1]. The upper limit on the small file depends on a particular tape technology and has tendency to grow. For now and foreseeable future the reasonable limitation is 500 MB. Today's large file is tomorrow's small file

Users can not always easily control the size of files they write to tapes, and many storage systems provide transparent aggregation of files for the user. Enstore does not provide this functionality and this project is intended to provide this. It is of particular interest to existing and upcoming Neutrino experiments, whose data files are typically small.

Another fact to take into considerations is that there are already many files with relatively small sizes stored in the Fermilab enstore tape systems. We are migrating from tapes with small files to tapes with much larger capacity, resulting in tapes with tens of thousands of small files on them. The access to such files can be quite slow and inefficient, tying up valuable resources. To optimize access and transfer rates for relatively small files we need to create a mechanism of packaging such files as a single entity (package), stored on tape, while at the same time permitting transparent access to each file in a package.

Packaging files before writing them to tape requires a disk buffer to aggregate files into a package. Unpacking files from their package retrieved from tape before they get delivered to users requires a disk cache. This disk space can not be requested on the user side because users may not be able to give up a part (sometimes substantial) of their disk space for packaging. This disk cache must be an internal to enstore , allowing to optimize packing / unpacking and delivery of individual files.

This cache/buffer will be used by multiple migrators/stagers to transfer files (packaged or not) between disks and tapes in a distributed environment. Files will arrive from client nodes to the caching system and will be stored on its disks. To provide a flexible environment for such files the caching system should provide access to each file in cache from any host involved in transfer, packaging, migration, staging, and unpacking. Thus the main requirement for such a cache is to provide a global access to any file, which can be achieved by using a clustered file system.

2 Structure of enstore caching system.

1 We consider an optimal file size the size, which provides the throughput not less than 90% of maximum

The structure of integrated caching system is illustrated in   fig 1. Such a system will easily scale by adding migrators and by expanding the clustered file system. The clustered file system sustain simultaneous high data throughput to multiple sources and destinations.

Enstore caching system is implemented with encp clients and disk movers. This implementation allows to reuse reliable data delivery mechanisms already incorporated into enstore, such as CRC calculations, internal retries, etc. The data caching system is shown in Fig. 1.

Communication between enstore caching system components in Fig.1 is supported by two protocols: Enstore UDP (EUDP) and AMQP. EUDP protocol is used because some components of enstore caching systems are standard enstore servers and clients. AMQP is used to communicate with components involving Policy Engine because it is one of its possible communication protocols. For integration of these 2 protocols the EUDP/AMQP Proxy Server is used. Its role is to receive messages in UDP format from existing enstore components and send to new components in AMQP and vice verse. More than one EUDP/AMQP can be configured in the system to scale the system throughput.

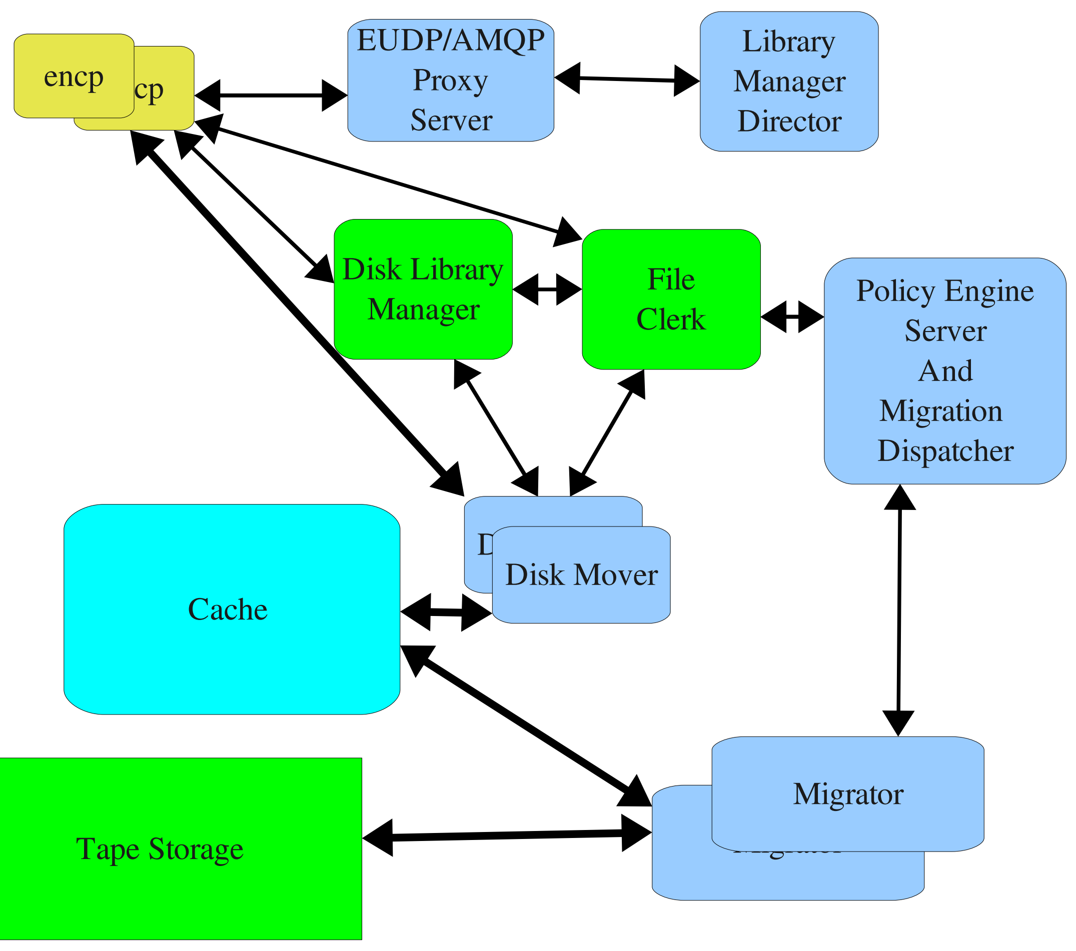


Fig 1. Structure of integrated data caching and tape system using encp and disk movers.

The Library Manager Director (LMD) functionality is to determine whether to send data to tape library directly or to cache first. Encp (client) sends to Library Manager Director request (ticket) containing library name from “library” tag in pnfs directory it tries to write file to. Library Manager Director has associated with it Policy Engine with set of rules defining selection of ether tape or disk library manager. The selection rules can be based on different parameters. These parameters are discussed in details later. The main is the file size. Each external encp client may contact LMD for write request if it uses a special option “--enable-redirection”. In the future the default behavior may get changed so that each encp will contact LMD.

Very important detail in this structure is that all disk movers are connected to the same clustered file system, so that they all can access any file in cache. Further we will address details of enstore caching system using encp clients and disk movers.

2.1 Hardware for cache

The enstore cache will be implemented as a clustered file system available for access by any migrator. It must be very reliable to reduce to the minimum possible corruption of data for files written to cache and awaiting their migration to tapes. It has been decided to use Oracle ZFS based disk system Advanced HPC File Server Nexenta OS (Open Solaris with Linus user level utilities with 63 TB of disk space, 48 GB memory and 10 GB Ethernet network interface.

2.2 Files and cache

2.2.1 File names in cache.

Files written into cache by disk mover have their path and names derived from the file pnfs id. Pnfs id consists of 36 hexadecimal digits. To prevent having too many files in a single directory in cache the pnfs file id maps to the corresponding file name according to algorithm:

```
file_id_hex = int("0x"+file_id, 16)
first = "%s"%((file_id_hex & 0xFFF) ^ ( (file_id_hex >> 24) & 0xFFF),)
second = "%s"%((file_id_hex>>12) & 0xFFF,)
path = os.path.join(root, first, second, file_id)
```

So for instance:

```
root = "/data_files"
file_id = "00001E9281CFB7054652B62737ED1ED3B3F6"
return value:
"/data_files/3816/3387/00001E9281CFB7054652B62737ED1ED3B3F6"
```

Thus each path will contain not more than 0xfff (4095) files. And files will be evenly distributed between different directories.

All files get written by disk mover into a temporary directory, with name unique for each disk mover:

/prefix/DN

- where DN is a disk mover name from enstore configuration

When disk mover completes write file transfer into this directory it immediately renames it according to a pnfs id name convention. Only one file can be in this temporary directory for each disk mover.

This approach guaranties that there will be no partially files on disk due to disk mover failures during file transfers (for write operations).

2.2.2 Package file.

Package file contains a files packaged according certain criteria defined by policy and a special file README.1ST with the following format:

List of cached files and original names

cached_file_path_1 file_name_in_the_name_space1 CRC1

cached_file_path_2 file_name_in_the_name_space2 CRC2

....

cached_file_path_N file_name_in_the_name_spaceN CRCN

This file makes package file self describing and allows to recover files from tape to their original locations in cases when enstore database is absent. These cases may be:

1. Loss of the database
2. Transfer of the tape to another system, which may be enstore or some other tape based storage system.

2.2.2.1 Package file name convention

Package files are written and read by encp client, so they are regular enstore and pnfs files.

Package files for archiving (to get written to tape) will be kept in the special archiving area.

The separate directory for a packaged file is needed to not allow a conflict between package file contents during packaging.

Staged package files will be kept in staging area

/<stage>

where <archive> and <stage> paths to archiving and staging areas.

Archiving and staging areas can be part of enstore cache or just local disk areas on Migrator nodes.

Package file path in the name space is:

<name_space_for_package>/<volume_family>/<tape>/package-<Migrator>-YYYY-mm-ddT%HH:%MM:%SSZ.tar,

where <Migrator> is the name of migrator, which wrote this file. Time stamp suffix is in accordance to ISO8601 time format.

2.2.2.2 Additional entries in file table of enstore DB

Additional entries in **file table of enstore DB** are:

packageId - character varying default '' - This entry is a bfid of a package file, indicates that the file is a part of package and can not be accessed as an individual file directly on tape.

PackageFileCounter integer default -1 – counter of active files in package (gets updated only for a package file).

PackageFileNumber integer default 0 – total number of files in a package.

Package can be deleted when ***PackageFileCounter=0.***

2.2.2.3 Cached file information

If file was written to cache it will have the following additional entries in **file table of enstore DB**:

cache_status character varying default ''
archive_status character varying default ''
cache_mod_time timestamp without time zone
archive_mod_time timestamp without time zone
where:

- cache_status – shows if file is in cache/migration/purge status. It can have the following values:
  - CREATED – file was written to cache
  - PURGING – file is being purged
  - PURGED – file was deleted in cache
  - STAGING_REQUESTED – staging of the file has commenced
  - STAGING – file is being staged to cache from tape
  - CACHED – file is in cache
  - FAILED – something wrong happened, requires investigation
  - None – default value
- archive_status – file migration/purge status. It can have the following values:
  - ARCHIVING – file is being written to tape. This state is useful for the recovery from failure
  - ARCHIVED – file was written to tape
  - FAILED – something wrong happened, requires investigation
  - None – default value
- cache_mod_time – time when cache_status has changed
- archive_mod_time – time when archive_status has changed

New table in file DB holds a status of files in transition – files_in_transition. This table contains files written to cache but not yet migrated to tape.

bfid character varying

file_status character varying default ''

cache_mod_time timestamp without time zone.

Special purge rules in PE Server define if a certain file needs to get purged.

2.3 Enstore Caching System Components.

In this section Enstore Section Components shown in Fig. 1 will be described.

2.3.1 UDP to AMQP proxy servers.

Each enstore UDP to AMQP proxy has two proxy servers: one to convert enstore udp messages to amqp messages and the other to convert amqp replies to enstore UDP reply tickets.

UDP2AMQ proxy server receives enstore UDP messages and uses its payload (enstore ticket) as *content* of AMQP message. The resulting qpid Message is sent to qpid target address defined in configuration.

2.3.1.1 UDP2AMQ Proxy Configuration.

Configuration may have one AMQP broker defined as:

```
configdict['amqp_broker'] = {
  'host':enstore_qpid_broker_host,
  'port':5672,
  }
```

'host' – host on which AMQP broker is running

'port' – AMQP broker communication port

Configuration dictionary may have entries for multiple proxy servers. Each entry has description as follows:

```
configdict['my_proxy.proxy_server'] = {
    'host': 'proxyhost.fnal.gov',
    'port' : 7700, # udp server port
    'udp_port': 7710,
    'target_addr':'udp_relay_test'
}
```

'host' – host on which 'my_proxy.proxy_server' is running

'port' - 'my_proxy.proxy_server' communication port as enstore server

'udp_port' – port on which proxy server transfers enstore UDP messages

'target_addr' – queue on which proxy server transfers AMQP messages.

2.3.1.2 UDP2AMQP Proxy Server enstore commands.

Udp2amqp proxy server is a regular enstore server, monitored by inquisitor and shown on System Status page and Servers page. It can be started and stopped just as other enstore servers using “enstore start/stop” command. It responds to the following commands:

```
[enstore@dmsen02 test_dir]$ enstore udp
Usage:
udp [ -ha --alive --help --retries= --timeout= --usage ] udp_proxy_server
-a, --alive prints message if the server is up or down.
-h, --help prints this message
--retries <ALIVE_RETRIES> number of attempts to resend alive requests
--timeout <SECONDS> number of seconds to wait for alive response
--usage prints short help message
```

2.3.2 Library Manager Configuration Changes.

Configuration dictionary for each Library Manager may have entry with new key `“use_LMD”` with value containing reference (key) configuration dictionary entry for LMD, e.g.

```
configdict['LTO3.library_manager'] = { …,'use_LMD': 'lmd_proxy',}
```

When `'use_LMD'` entry is present, encp shall contact specified LMD to get configuration entry for Library Manager to be used for transfer. When entry is not present encp acts in a “classical” way.

2.3.3 Library Manager Director.

The Library Manager Director (LMD) functionality is to determine shall client send data directly to tape library or use file aggregation in cache. External encp clients contact LMD first. LMD decides destination of the transfer: tape or enstore cache based on the set of policy rules defined to perform selection of tape or disk based Library Manager. The selection rules can be based on different parameters most importantly file size.

The LMD is a regular enstore server, monitored by inquisitor and shown on System Status page and

Servers page. It can be started and stopped just as other enstore servers using “enstore start/stop” command.

2.3.3.1 Library Manager Director configuration.

Library Manager Director configuration is as:

```
configdict['lm_director'] = {
    'host': 'pmig01.fnal.gov',
    'port': 5602,
    'logname':'LMDSRV',
    'queue_in': 'udp_relay_test',
    'udp_proxy_server': 'lmd.udp_proxy_server',
    'policy_file': "/home/enstore/policy_files/lmd_policy.py"
    }
```

'host' - host on which “lm_director” is running

'port' - “lm_director” communication port as enstore server

'logname' - “lm_director” log name

'udp_proxy_server' – proxy server with which “lm_director” communicates

'queue_in' – AMQP queue name on which “lm_director” communicates with proxy server

'policy file' – file containing policies for a given LMD

2.3.3.2 Policy File Format

Library Manager Director and implemented in Python Policy Engine Server / Migration Dispatcher use the same policy file, although it can be different.

The policy file is a Python dictionary and has a following format:

```
{Library_Manager1: {policy1:
                    {'rule':
                         {'storage_group': 'G1',
                          'file_family': 'F1',
                          'wrapper':'cpio_odc'
                         }
                    'minimal_file_size': 2000000000L
```

```
                        'min_files_in_pack': 100,
                        'max_waiting_time': 300,
                        'resulting_library': 'new_library'
                        }
                   {policy2:
                        {
                        .....
                        },
                  ....
                 },
Library_Manager2: { policy1:
                        ....
                    }
```

Here is an example and explanation

```
 'LTO3.library_manager':{1: {'rule': {'storage_group': 'G1',
                              'file_family': 'F1',
                              'wrapper':'cpio_odc'
                             }
                 'minimal_file_size': 2000000000L
                 'min_files_in_pack': 100,
                 'max_waiting_time': 300,
                 'resulting_library': 'disk_library'
                 }
```

'minimal_file_size' - if file is less than this size the file will be aggregated

'min_files_in_pack' - minimal number of files in package,

if total size of files to be aggregated is less than minimal_file_size

and number of files >= min_files_in_pack then files will get packaged

'max_waiting_time' - if time of collection of files for a package exceeds this value (sec), the files will get packaged

If request comes from encp with library LTO3.library_manager and it satisfies this rule and minimal_file_size conditions (less than the minimum) it will be sent to ' disk_library'.

2.3.3.3 Library Manager Director enstore commands

Library Manager Director is a regular enstore server, monitored by inquisitor and shown on System Status page and Servers page. It can be started and stopped just as other enstore servers using "enstore start/stop" command. It responds to the following commands:

[enstore@dmsen02 test_dir]$ enstore lmd

Usage: lmd [OPTIONS]...

-a, --alive

prints message if the server is up or down.

--do-alarm <DO_ALARM> turns on more alarms

(snip ...)

--load load a new policy file

--retries <ALIVE_RETRIES> number of attempts to resend alive requests

--show - print the current policy in python format

--timeout <SECONDS> number of seconds to wait for alive response

--usage - prints short help message

The options of interest are –load and –show.

2.3.3.4 Encp Interaction with Library Manager Director.

The new encp switch allows users to select whether they want to use the file aggregation feature: ***enable-redirection.*** The default value of this switch is to not use Library Manager Director to select a library manager. If this switch is specified the encp will try to send a request to a Library Manager Director.

User side encp client extracts library name from `“library”` tag in pnfs directory (name space) where it tries to write file to. For enstore pnfs tags see "**The Enstore and dCache User's Guide. Chapter 4: PNFS Namespace 4-1**". Enstore configuration file will have entry specifying if caching is enabled for concrete tape Library Manager. The configuration dictionary entry for Library Manager may have entry `“use_LMD”`. For backward compatibility, if there is no such entry encp acts in a old way, encp does not contact LMD and it contacts Library Manager directly. When `“use_LMD”` entry is present in configuration its value contains key in configuration dictionary for LMD. Encp contacts specified LMD to get configuration entry for Library Manager to be used for transfer. Encp sends ticket to LMD in the same format it usually sends to Library Manager. The ticket contains library name from “library” tag in pnfs directory (name space).

The choice of caching Library Manager is described by LMD Policy Engine rules. As a simple case, each original tape Library Manager has corresponding Caching Library Manager. LMD make decision to cache or not to cache and selects one of LMs from the pair.

LMD receives ticket from encp, processes the ticket in policy engine, modifies ticket field ***ticket[“vc”][“library”]*** if necessary and then sends ticket as a reply back to encp.

2.3.4 Modified File Clerk

Existing File Clerk (FC) was modified to notify Policy Engine (PE) Server when files are created in cache and when read file is requested. For the communications with PE Server file clerk uses AMQP API, described in “Messaging HLD”. The specific of enstore file write operation is such that the file may not get considered as written into enstore even if mover successfully completed data transfer. File is considered as written into enstore when enstore client (encp) sets the pnfsId of written file in enstore file database by calling a corresponding File Clerk Client (FCC) method, which in turn sends a message to FC. When this is done File Clerk sends CACHE_WRITTEN event to PE Server. A special process (thread) in File Clerk will set timeouts when File Clerk gets a new_bit_file request from disk mover. If set_pnfsid request does not arrive from encp for the corresponding bfId, File Clerk raises an alarm and makes an entry into the list of suspect files on disk. This is needed for the subsequent cleanup of not completed file writes.

For read requests Library Manager checks if requested file is on disk and if not it sends an ***Open*** request to File Clerk, which sends CACHE_MISS event to PE Server.

More details about File Clerk modifications can be found in File Clerk and enstore DB modifications

2.3.4.1 Additional File Clerk Enstore Commands

The following command options were added:

--children <BFID>	find all children of the package file
--replay	replay cache written events – replays events for files_in_transition table
--package	Force printing package files and non-packaged files (used along with --list)
--pkginfo	Force printing information about package_id archive/cache status (used along with --list)

2.3.5 Policy Engine Server.

Policy Engine Server (PE Server) collects events from other servers (particularly from File Clerk) combines them into the request lists and submits them to Migration Dispatcher. This is why it is implemented as a part of the process running both PE Server and Migration Dispatcher. The communication between PE Server and Migration Dispatcher is done via shared memory. The events destined for PE Server are described in the corresponding “Messaging HLD” document. PE Server receives event from File Clerk specifying that the **new** file arrived into cache or the file written to tape needs to get staged from it. PE Server has 3 types of file lists:

1. Archive Lists. List of files to be written to tape.
2. Stage Lists. Lists of files to be staged from tape(s) to cache.
3. Purge Lists. Lists of files to be purged in cache.

Lists may be groups of files belonging to a certain storage group, file family, directory, having certain size limits, time in cache, etc. Dispatcher has

- filling – list is being populated with new files
- full – list is full
- work – list is sent for the execution
- done – execution was completed successfully
- failed – execution failed.

When policy rule is satisfied for a certain list it gets transferred to Migration Dispatcher.

2.3.5.1 File List Format.

File list format is as follows:

File list ID - unique file list id.

list_item_1[, list_item_2, ... , list_item_N]

where list_item_*i* is:

enstore bit file ID(BFID) – assigned when file gets written into enstore disk cache

name space file id (pnfs Id) – name space (pnfs) id. This allows to avoid contacting file clerk to fetch a file on disk

file path – complete file path. This allows to avoid contacting file clerk.

tape library list – information from pnfs library tag

file CRC

All information about requested file can be obtained from enstore file database referring by BFID.

Tape library list is needed to identify to tape(s) from what library (or libraries for multiple copy) the files will be written of from what tape in what library it will (or may be) staged. The operation type (archive, stage, purge) performed on the list is specified in the message sent to the server.

2.3.6 Migration Dispatcher.

Migration Dispatcher (MD) receives file lists from Policy Engine Server over shared memory and distributes file lists between Migrators or purges files depending on the operation specified in the list. When migrator replies with result of the work, MD sends corresponding event to Policy Engine Server.

2.3.6.1 Migration Dispatcher Configuration

Migration Dispatcher configuration is described in enstore configuration file, presents a python dictionary, and contains the following attributes:

'host': - MD host (string)

'port': - MD port (string)

'logname': - MD log name in enstore log (string)

'norestart':' - automatic restart flag (string)

2.3.7 Python implementation of Policy Engine Server and Migration Dispatcher – Dispatcher.

Python implementation of Policy Engine Server and Migration Dispatcher has both these components running in the same process, because they share a lot of information. They run in separate threads.

The Dispatcher is a regular enstore server, monitored by inquisitor and shown on System Status page and Servers page. It can be started and stopped just as other enstore servers using “enstore start/stop” command. Its configuration is as:

```
configdict['dispatcher'] = {
    'host': 'pmig01.fnal.gov',
    'port': 5603,
    'logname':'DISP',
    'queue_work': 'policy_engine',
    'queue_reply': 'file_clerk',
    'migrator_work': 'migrator',
    'migrator_reply': 'migrator_reply',
'policy_file': "/home/enstore/policy_files/lmd_policy.py",
'max_time_in_cache': 600,
'purge_watermarks':(.8, .4),
}
```

'host': - sever host
'port' - server port

'logname' – server log name

'queue_work': - events (from file clerk) come to this queue

'queue_reply': - replies (if needed) are sent on this queue

'migrator_work': - request lists are sent to this queue

'migrator_reply': - replies from migrators come to this queue

'policy_file': - policy file (same as for Library Manager Director).

'max_time_in_cache': 600 – purge file if it was written max_time_in_cache ago

'purge_watermarks':(.8, .4) – start purging staged files if occupied space is more than .8*Total,
stop purging files if occupied space is less than .4*Total

Dispatcher responds to the following commands:

```
[enstore@pmig01 src]$ enstore disp --help
 Usage:
     disp [OPTIONS]...


  -a, --alive             prints message if the server is up or down.
     --delete-work <DELETE_WORK>  delete list from migration pool identified
                          by its id
     --do-alarm <DO_ALARM>  turns on more alarms
     --do-log <DO_LOG>     turns on more verbose logging
     --do-print <DO_PRINT>  turns on more verbose output
     --dont-alarm <DONT_ALARM>  turns off more alarms
     --dont-log <DONT_LOG>  turns off more verbose logging
     --dont-print <DONT_PRINT>  turns off more verbose output
     --get-queue          print content of pools
  -h, --help              prints this message
     --id <ID>            get information about specific ID in migration pool.
     --load               load a new policy file
     --retries <ALIVE_RETRIES>  number of attempts to resend alive requests
     --show               print the current policy in python format
     --start-draining     start draining write requests
     --timeout <SECONDS>  number of seconds to wait for alive response
     --usage              prints short help message
  -v, --verbose           verbose output. Used with –get-queue
```

The options of interest are load, show (same as for Library Manager Director), get-queue, id, delete-work, start-draining

Here is an example output of “enstore disp –get” command:

```
[enstore@dmsen03 ~]$ enstore disp --get
Pool cache_missed Pool size 0
Pool cache_purge Pool size 0
Pool cache_written Pool size 1
Pool migration_pool Pool size 6
id e2db31c4-61bc-4cbd-9a53-64f484fb9b8c policy LTO3GS.ANM.FF1.cpio_odc. list length 19 type CACHE_WRITTEN
time_qd Tue Mar 27 14:40:24 2012
id 96ea70d9-30ed-4c3d-a7de-01b4ae33138e policy LTO3GS.ANM.FF1.cpio_odc. list length 9 type CACHE_WRITTEN
time_qd Tue Mar 27 14:34:27 2012
id ba71a926-f702-4d5f-a8db-b0661f256b5b policy LTO3GS.ANM.FF1.cpio_odc. list length 21 type CACHE_WRITTEN
time_qd Tue Mar 27 14:31:39 2012
id 1332877307_0 policy Purge_1332877307_0 list length 6 type MDC_PURGE time_qd Tue Mar 27 14:41:47 2012
id cda34e66-6031-448c-8a63-27b813b858c7 policy LTO3GS.ANM.FF1.cpio_odc. list length 18 type CACHE_WRITTEN
time_qd Tue Mar 27 14:42:26 2012
id 04ce6bbb-6732-4b3e-842d-9656b596e784 policy LTO3GS.ANM.FF1.cpio_odc. list length 22 type CACHE_WRITTEN
time_qd Tue Mar 27 14:32:23 2012
```

Here is a example of verbose output:

```
 'cache_purge': {},
'cache_written': {'LTO3GS.ANM.FF1.cpio_odc.': id=27ba6428-c548-484a-99dc-14f9e0997c5d
name=LTO3GS.ANM.FF1.cpio_odc. type=CACHE_WRITTEN content [{'libraries': ['LTO3GS'], 'path':
'/pnfs/data2/file_aggregation/LTO3/moibenko/dmsen06/torture_test/tape/dmsen03/38/dmsen03_7e1e0bdc784311e1a6040
030487c224e.data', 'bfid': 'GCMS133287747300000', 'nsid': '00007658E5192007424CB813BDA090EDA855',
'complete_crc': 3228300499}, {'libraries': ['LTO3GS'], 'path':
'/pnfs/data2/file_aggregation/LTO3/moibenko/dmsen06/torture_test/tape/dmsen03/35/dmsen03_7e5dfae4784311e1961b0
030487c224e.data', 'bfid': 'GCMS133287748200000', 'nsid': '0000A87F4BEEF4174DD18C7EFFC3B7A59937',
'complete_crc': 3548739519}, {'libraries': ['LTO3GS'], 'path':
'/pnfs/data2/file_aggregation/LTO3/moibenko/dmsen06/torture_test/tape/dmsen03/63/dmsen03_7fe45d36784311e186e10
030487c224e.data', 'bfid': 'GCMS133287748300000', 'nsid': '00008A59D1665648461F80577323B427F95C',
'complete_crc': 2683267258}, {'libraries': ['LTO3GS'], 'path':
'/pnfs/data2/file_aggregation/LTO3/moibenko/dmsen06/torture_test/tape/dmsen03/32/dmsen03_80dd3db6784311e1b4a8
0030487c224e.data', 'bfid': 'GCMS133287749800000', 'nsid': '00006472BDD7B7D642DCB3B7543DA3537C07',
'complete_crc': 2603776297}, {'libraries': ['LTO3GS'], 'path':
'/pnfs/data2/file_aggregation/LTO3/moibenko/dmsen06/torture_test/tape/dmsen03/30/dmsen03_81eab3aa784311e1a98d0
030487c224e.data', 'bfid': 'GCMS133287750000000', 'nsid': '0000C49CA019CB44458C8C381A5BE54C2E15',
'complete_crc': 3652691175}, {'libraries': ['LTO3GS'], 'path':
'/pnfs/data2/file_aggregation/LTO3/moibenko/dmsen06/torture_test/tape/dmsen03/41/dmsen03_834829b2784311e18aa00
030487c224e.data', 'bfid': 'GCMS133287751500000', 'nsid': '0000680410435A834296B3A5B90438D411D5',
'complete_crc': 400400435}, {'libraries': ['LTO3GS'], 'path':
'/pnfs/data2/file_aggregation/LTO3/moibenko/dmsen06/torture_test/tape/dmsen03/43/dmsen03_84635e52784311e185780
030487c224e.data', 'bfid': 'GCMS133287751700000', 'nsid': '0000B34B12E1C9E04EF597214BF3F4AC15CB',
'complete_crc': 400400435}, {'libraries': ['LTO3GS'], 'path':
'/pnfs/data2/file_aggregation/LTO3/moibenko/dmsen06/torture_test/tape/dmsen03/15/dmsen03_857db242784311e190930
030487c224e.data', 'bfid': 'GCMS133287751900000', 'nsid': '0000C33037FCA2A2475BB96B4F12E9A69699',
'complete_crc': 175899208}, {'libraries': ['LTO3GS'], 'path':
'/pnfs/data2/file_aggregation/LTO3/moibenko/dmsen06/torture_test/tape/dmsen03/14/dmsen03_86376930784311e184bb0
030487c224e.data', 'bfid': 'GCMS133287752300000', 'nsid': '0000346CABC0DD034784B685B43751410482',
'complete_crc': 647363521}, {'libraries': ['LTO3GS'], 'path':
'/pnfs/data2/file_aggregation/LTO3/moibenko/dmsen06/torture_test/tape/dmsen03/44/dmsen03_87662332784311e1b4b80
030487c224e.data', 'bfid': 'GCMS133287752400000', 'nsid': '00004E3E06C90891479F80AA6CCAC37D0689',
'complete_crc': 3175043570}, {'libraries': ['LTO3GS'], 'path':
'/pnfs/data2/file_aggregation/LTO3/moibenko/dmsen06/torture_test/tape/dmsen03/95/dmsen03_885ee044784311e1aed40
030487c224e.data', 'bfid': 'GCMS133287752800000', 'nsid': '00001362D9957B8B497DB7040990C4DAD51E',
'complete_crc': 3908708536}, {'libraries': ['LTO3GS'], 'path':
'/pnfs/data2/file_aggregation/LTO3/moibenko/dmsen06/torture_test/tape/dmsen03/66/dmsen03_89b4d96c784311e194b80
030487c224e.data', 'bfid': 'GCMS133287752900000', 'nsid': '00007F0592EFAD0E47D0BB572F993F8A7D75',
'complete_crc': 1639662650}, {'libraries': ['LTO3GS'], 'path':
'/pnfs/data2/file_aggregation/LTO3/moibenko/dmsen06/torture_test/tape/dmsen03/88/dmsen03_8a5734a0784311e188a90
030487c224e.data', 'bfid': 'GCMS133287753300000', 'nsid': '0000844BDA7422B24FC9A3F4DC685901CEE2',
'complete_crc': 269296325}, {'libraries': ['LTO3GS'], 'path':
'/pnfs/data2/file_aggregation/LTO3/moibenko/dmsen06/torture_test/tape/dmsen03/53/dmsen03_8b2bf35c784311e183560
030487c224e.data', 'bfid': 'GCMS133287753300001', 'nsid': '0000D8E536712B7341CA89F14399B93FC8A2',
'complete_crc': 2669110146}, {'libraries': ['LTO3GS'], 'path':
'/pnfs/data2/file_aggregation/LTO3/moibenko/dmsen06/torture_test/tape/dmsen03/47/dmsen03_8de9be12784311e1a4e00
030487c224e.data', 'bfid': 'GCMS133287753900000', 'nsid': '000078C334FDB740443ABD1FB654079DB704',
'complete_crc': 4098067407}, {'libraries': ['LTO3GS'], 'path':
'/pnfs/data2/file_aggregation/LTO3/moibenko/dmsen06/torture_test/tape/dmsen03/67/dmsen03_8db911cc784311e184dd0
```

```
030487c224e.data', 'bfid': 'GCMS133287753900001', 'nsid': '00000174FB37557348678DC8CDD00DE2F825',
'complete_crc': 650822416}, {'libraries': ['LTO3GS'], 'path':
'/pnfs/data2/file_aggregation/LTO3/moibenko/dmsen06/torture_test/tape/dmsen03/75/dmsen03_8f6df262784311e19c0e0
030487c224e.data', 'bfid': 'GCMS133287754600000', 'nsid': '0000D1D2C4964BA34E36829DE799A381F702',
'complete_crc': 4187926220}, {'libraries': ['LTO3GS'], 'path':
'/pnfs/data2/file_aggregation/LTO3/moibenko/dmsen06/torture_test/tape/dmsen03/94/dmsen03_904121d2784311e1a2020
030487c224e.data', 'bfid': 'GCMS133287754900000', 'nsid': '00004B7B12525A4E4037A61E89969A6A4338',
'complete_crc': 3898010988}, {'libraries': ['LTO3GS'], 'path':
'/pnfs/data2/file_aggregation/LTO3/moibenko/dmsen06/torture_test/tape/dmsen03/42/dmsen03_92acfed2784311e1b9e00
030487c224e.data', 'bfid': 'GCMS133287756300000', 'nsid': '0000F1A65780D87448BEAA43F7F9386D3100',
'complete_crc': 3393179417}]},

 'migration_pool': {'04ce6bbb-6732-4b3e-842d-9656b596e784': {'id': '04ce6bbb-6732-4b3e-842d-9656b596e784',

                              'list': [{'bfid': 'GCMS133287669900000',

                                  'complete_crc': 647363521,

                                  'libraries': ['LTO3GS'],

                                  'nsid': '00008B73EABBAC7B4C55BA9C416A86C46DA2',

                                  'path':
'/pnfs/data2/file_aggregation/LTO3/moibenko/dmsen06/torture_test/tape/dmsen03/14/dmsen03_2ad5a19c784311e184bb0
030487c224e.data'},

                                 {'bfid': 'GCMS133287670100000',

                                  'complete_crc': 3175043570,

                                  'libraries': ['LTO3GS'],

                                  'nsid': '00003DCE96614D794BC88E9BEF5D44437FD3',

                                  'path':
'/pnfs/data2/file_aggregation/LTO3/moibenko/dmsen06/torture_test/tape/dmsen03/44/dmsen03_2c110312784311e1b4b80
030487c224e.data'},

                                 {'bfid': 'GCMS133287670300000',

                                  'complete_crc': 3908708536,

                                  'libraries': ['LTO3GS'],

                                  'nsid': '0000476A21DB3D42461599D185DBBD8966C7',

                                  'path':
'/pnfs/data2/file_aggregation/LTO3/moibenko/dmsen06/torture_test/tape/dmsen03/95/dmsen03_2cb46a3e784311e1aed40
030487c224e.data'},

                                 {'bfid': 'GCMS133287670500000',

                                  'complete_crc': 1639662650,

                                  'libraries': ['LTO3GS'],

                                  'nsid': '000067BD428141FF4E83BADE60525D3EE558',

                                  'path':
```

```
'/pnfs/data2/file_aggregation/LTO3/moibenko/dmsen06/torture_test/tape/dmsen03/66/dmsen03_2e004624784311e194b80
030487c224e.data'},

.....

sen03_3ff6fc9c784311e1aba20030487c224e.data'},

                                    {'bfid': 'GCMS133287676300000',

                                     'complete_crc': 1095276467,

                                     'libraries': ['LTO3GS'],

                                     'nsid': '0000E4CAAA521864407A9C44F23A0A31864B',

                                     'path':
'/pnfs/data2/file_aggregation/LTO3/moibenko/dmsen06/torture_test/tape/dmsen03/68/dmsen03_413540fa784311e1a6860
030487c224e.data'}],

                                'policy': 'LTO3GS.ANM.FF1.cpio_odc.',

                                'time_qd': 'Tue Mar 27 14:34:27 2012',

                                'type': u'CACHE_WRITTEN'},

.....

}}}
```

2.3.8 Migrator.

Migrator is responsible for moving files between Enstore disk cache and tapes in both directions. When files are being written to Enstore. Migrator receives a list of files from Migration Dispatcher. If this list is a list of files to be written to tape, Migrator aggregates these files into container and writes this container to tape. It then notifies MD that the file was written to tape, for each file in the list. If list, received from Migration Dispatcher is a list of files to be staged from tape Migrator stages requested files from tape to cache. Migrator will also stage files that happen to be in the package along with requested files.

2.3.8.1 Migrator configuration and enstore commands.

Migrator is a regular enstore server, monitored by inquisitor and shown on System Status page and Servers page. It can be started and stopped just as other enstore servers using “enstore start/stop” command. Migrator configuration is described in enstore configuration file, presents a python dictionary, and contains the following attributes:

'host': - migrator host name (string)

'port': - migrator port number (integer)

'logname': - migrator name in the enstore log file (string)

'migration_dispatcher' – pointer to migration dispatcher structure in enstore configuration (keyword)

'data_area' – disk area where files are stored 'archive_area' – disk area where archive is created during data archiving (string)

'stage_area' – disk area where archived files get staged from tape (string)

'tmp_stage_area' – disk area where staged files are temporarily stored (string)

'packages_dir' - directory in name space for packages

'dismount_delay' – delay for dismounting tape

'check_written_file' - if greater than 0, then randomly check files written using this number as the mean (default value:0 - don't check)

Migrator responds to the following enstore commands:

```
[enstore@enmvr005 ~]$ enstore mig
Usage:
      mig [OPTIONS]... migrator_name
  -a, --alive             prints message if the server is up or down.
   -h, --help              prints this message
      --offline           offline migrator
      --retries <ALIVE_RETRIES>  number of attempts to resend alive requests
      --status            print migrator status
      --timeout <SECONDS>   number of seconds to wait for alive response
      --usage             prints short help message
```

Here is an example output of migrator status:

```
[enstore@dmsen03 ~]$ enstore mig --status M1
{'status': ('ok', None), 'internal_state': 'WRITING_TO_TAPE', 'work': 'get_status',
'time_in_internal_state': 5.0869381427764893, 'state': 'ARCHIVING', 'time_in_state':
84.499337911605835}
```

Currently migrator can be in the following states:

Intermediate:

PURGING – in process of being purged in cache

STAGING - stagung from tape to cache

STAGING_REQUESTED – staging from tape was requested

ARCHIVING - in process of being packaged and written to tape.

Final:

ARCHIVED - on tape

PURGED - purged from cache

CACHED - in cache

FAILED – failed, needs special attention

Internal states:

IDLE

PACKAGING - packaging file before sending to tape

UNPACKAGING - unpackaging files from package, staged from tape

CHECKING_CRC – checking crc of packaged files before sending to tape

WRITING_TO_TAPE

REGISTERING_ARCHIVE – registering files as ARCHIVED

CLEANING - cleaning files after operation on them was completed.

PREPARING_READ_FROM_TAPE – sate before reading from tape in STAGING

READING_FROM_TAPE - reading from tape in STAGING

Migrator “offline” command option is like the corresponding move option. When issued it causes migrartor to finish its current work and exit (this is different from mover behavior).

3 Write and read request processing.

Write and read request processing is graphically explained in “File Aggregation in Enstore” (CS-doc-4596-v1)

Here it will be described what is presented on corresponding pages in this document on page 10 (write) and 11 (read)

3.1 Write request processing of files written via cache

1. Encp sends write request to UDP proxy server in enstore UDP format.
2. UDP proxy server converts message to AMQP format and sends it Library Manager Director (LMD).
3. LMD defines the library manager to send a request to, according to policy and sends reply to UDP proxy server with modified (if required by policy) ticket[“vc”][“library”] and the name of the original library in ticket[“vc”][“original_library”]
4. UDP proxy server converts it back to enstore UDP format and sends it to encp.

5. Encp sends write request to Library Manger (LM) defined by ticket[“vc”][“library”]
6. Disk Mover (DM) sends a request for work to LM. LM sends write_to_hsm work to DM
7. DM transfers data from encp
8. To disk.
9. DM creates new bit file calling new_bit_file method of File Clerk Client. File Clerk creates bfId, sets cache_status to CREATED, makes entry in file_in_transition table and starts waiting for create_pnfsId call from encp
10. Encp completes file operations and calls create_pnfsId FC Client method
11. FC sets pnfsId, sets cache_status to CACHED, and sends CACHE_WRITTEN event to PE Server
12. PE Server creates lists of of events grouped according policy. The policy can be “group all files with certain volume family if the size of the individual file is less than specified”, “immediately migrate the file”, etc. When list fills in (satisfies fill in criteria, such as size of all files in the list) it gets handed over to Migration Dispatcher. Migration dispatcher sends this list to common queue and. Then it moves the list into active lists internally.
13. Migrator pulls list from common queue, packs files into package if necessary, sets all files cache_status to ARCHIVING
14. Migrator sends the package to tape using encp. The package gets written to tape according to pnfs tags, common to files in the package (as described in 2.2.2.1). Migrator sets statuses of all packaged files to ARCHIVED and removes corresponding bfids from files_in_transition in enstore DB

3.2 Read request processing.

1. Encp gets the library manager from the file's record in enstore DB (this is a standard way for encp, no changes are needed). It then send a request to library manager. If request comes to tape Library Manager, it gets sent to tape mover (when request for work comes from the mover). It then gets transferred to encp (client). If request comes to disk Library Manager, it checks whether the file is in cache using File Clerk information contained in the incoming request. If file is in cache (cache_status == CACHED) the request gets sent to disk mover (when request for work comes from the mover). It then gets transferred to encp (client) – skip to step 10. If file is not in cache and its cache_status != CACHED then there could be the following scenarios:
   1) cache_status == STAGING: This means that the file is being processed by one of migrators or is a part of package, one of which files is being processed by migrator. If disk LM can not find this file in at_movers list it moves this file in on_hold list.
      a) When migrator completes file stage from tape it sets cache_status = CACHED
      b) It then sends the stage confirmation to disk mover if there is one, waiting for a file, and to Disk Library Manager

2) cache_status == purged: request gets sent to disk mover in this case:

2. Disk Library Manager sends open_bitfile request for the bfid of a package files containing requested file.
3. When disk mover requests for the work Disk LM sends work for the requested file to it. Disk mover switches to state SETUP and waits until cache_status of requested file becomes STAGED. This may tape a long time as this process includes waiting in the queue of Tape Library Manager and reading from tape.
4. File clerk sends CACHE_MISSED event to Dispatcher
5. File Clerk sets cache_status to STAGING
6. Dispatcher sends stage request to the common migrator queue.
7. Migrator stages package file from tape.
8. Migrator unpacks package and sets cache_status to CACHED for all files in the package which status is not CACHED.
9. Mover detects this change and transfers a file from cace
10. To client encp

3.3 Purge file from cache

When cache fills in the files get purged in it. If separate caches are used for writes and reads the files get purged from write cache as soon as the maximum time in cache expires. This parameter is configurable and is defined in “Python implementation of Policy Engine Server and Migration Dispatcher – Dispatcher” configuration.

For files staged into read cache files will be purged if the high watermark in read cache is passed based on FIFO policy. The purge stops if the low watermark is passed (see configuration parameters in “Python implementation of Policy Engine Server and Migration Dispatcher – Dispatcher”.

Purge is done by migrators.

4 Administration.

4.1 Configuring enstore cache components.

Enstore cache comprises Clustered Disk Cache and enstore cache servers.

4.1.1 Clustered Disk Cache

Clustered Disk Cache is currently implemented as nfs mounted ZFS appliance. It is running on **pagg01.** By convention it has 3 cache areas:

/volumes/aggwrite/cache - write cache

/volumes/aggread/cache - read (stage cache)

/volumes/aggpack/cache - packaging cache where files written into write cache get packaged and written to tape.

These areas need to be nfs mounted on hosts running disk movers and migrators, as well as on the server where enstore web server runs (*srv2n).

Currently we have 2 hosts for enstore disk movers and migrators: **pmig01, pmig02.**

The mount points in **/etc/fstab** are:

```
pagg01:/volumes/aggwrite/cache  /volumes/aggwrite/cache nfs4 rw,hard,intr,bg,rsize=32768,wsize=32768 0 0
pagg01:/volumes/aggread/cache  /volumes/aggread/cache nfs4 rw,hard,intr,bg,rsize=32768,wsize=32768 0 0
pagg01:/volumes/aggpack/cache  /volumes/aggpack/cache nfs4 rw,hard,intr,bg,rsize=32768,wsize=32768 0 0
```

/etc/idmapd.conf must have

```
Domain = fnal.gov
```

4.1.2 AMQP Broker

AMQP broker is a part of enstore cache messaging system. It comes as part of enstore_cache rpm and starts automatically on boot. It has a corresponding entry in enstore configuration file. The example of broker configuration is:

```
configdict['amqp_broker'] = {
   'host':enstore_qpid_broker_host,
   'port':5672,
  }
```

4.1.3 Enstore Cache Servers.

Enstore cache servers are:

UDP Proxy Server, Library Manager Director, Dispatcher, Migrators.

The examples of configuration for these servers are:

UDP Proxy Server

```
configdict['lmd.udp_proxy_server'] = {
   'norestart':'INQ',
   'host': 'pmig01.fnal.gov',
   'port': 5601,
   'udp_port' : 7710,
   'target_addr':'udp_relay_test',
   }
```

Library Manager Director

```
configdict['lm_director'] = {
   'norestart':'INQ',
   'host': 'pmig01.fnal.gov',
   'port': 5602,
   'logname':'LMDSRV',
   'queue_in': 'udp_relay_test',
   'udp_proxy_server': 'lmd.udp_proxy_server',
   'policy_file': "/home/enstore/policy_files/lmd_policy.py"
   }
```

Dispatcher

```
configdict['dispatcher'] = {
   'norestart':'INQ',
   #'host': 'dmsen02.fnal.gov',
   'host': 'pmig01.fnal.gov',
   'port': 5603,
   'logname':'DISP',
   'queue_work': 'policy_engine',
   'queue_reply': 'file_clerk',
   'migrator_work': 'migrator',
   'migrator_reply': 'migrator_reply',
   'policy_file': "/home/enstore/policy_files/lmd_policy.py",
   'max_time_in_cache': 600,
   'purge_watermarks':(.8, .4),
   }
```

Migrator

```
configdict['M1.migrator'] = {
   #'host':'dmsen02.fnal.gov',
   #'host':'enmvr007',
   'host':'pmig01',
   'port': 5610,
   'logname': "M1MG",
   'packages_dir':migrator_pack_dir,
   'data_area': write_cache_area,
   'archive_area': archive_area,
   'stage_area': stage_area,
   'tmp_stage_area': tmp_stage_area,
   'norestart':'INQ',
   'migration_dispatcher':'dispatcher',
   'dismount_delay': 2,
   'check_written_file': mvr_check_f,
   'aggregation_host': nfs_host,
   'tar_blocking_factor':tar_blocking_factor,
   }
```

All configurations were described in chapter 2.3

4.2 Starting enstore cache components

4.2.1.1 AMQP broker

AMQP broker starts on boot. It can also be started by the following command (under root account):

```
[root@pmig01]# /etc/init.d/qpid_broker start
```

Checking if the broker runs:

```
[root@pmig01 init.d]# ps auxww | grep qpid
root     3516  0.0  0.0  61184   760 pts/2    S+   11:09   0:00 grep qpid
root     3769  0.1  0.0 297072 10552 ?        Ssl  Mar26   3:42 /opt/enstore_cache/qpid/sbin/qpidd -d
```

4.2.2 Enstore Cache Servers.

Enstore cache servers start by "enstore start ..." command as any other enstore server (under enstore account). The "enstore restart ..." command also works for enstore cache servers.

"EPS" and "Enstore EPS ..." commands can be used to check if the processes associated with enstore case servers run:

```
[enstore@pmig01 src]$ EPS
enstore  22578  0.0  0.0 154092 22578    1 ?      S    Mar26 00:00:00 /usr/bin/postmaster -p 5432 -D /var/lib/pgsql/data
enstore  22586  0.0  0.0 154208 22586    1 ?      Ss   Mar26 00:00:01 postgres: writer process
enstore  22587  0.0  0.0 154092 22587    1 ?      Ss   Mar26 00:00:00 postgres: wal writer process
enstore  24166  0.0  0.1 306428 24166    8 pts/4  Sl   09:54 00:00:00 python /opt/enstore_cache/sbin/dispatcher
enstore  24166  0.0  0.1 306428 24173    8 pts/4  Sl   09:54 00:00:00 python /opt/enstore_cache/sbin/dispatcher
enstore  24166  0.0  0.1 306428 24174    8 pts/4  Sl   09:54 00:00:00 python /opt/enstore_cache/sbin/dispatcher
enstore  24166  0.0  0.1 306428 24175    8 pts/4  Sl   09:54 00:00:00 python /opt/enstore_cache/sbin/dispatcher
enstore  24166  0.0  0.1 306428 24176    8 pts/4  Sl   09:54 00:00:00 python /opt/enstore_cache/sbin/dispatcher
enstore  24166  0.0  0.1 306428 24177    8 pts/4  Sl   09:54 00:00:00 python /opt/enstore_cache/sbin/dispatcher
enstore  24166  0.0  0.1 306428 24178    8 pts/4  Sl   09:54 00:00:00 python /opt/enstore_cache/sbin/dispatcher
enstore  24166  0.0  0.1 306428 24179    8 pts/4  Sl   09:54 00:00:00 python /opt/enstore_cache/sbin/dispatcher
enstore   7378  0.0  0.0 148892  7378    1 ?      S    Mar26 00:00:00 python /opt/enstore_cache/sbin/lm_director
enstore   7385  0.0  0.0 159464  7386    2 ?      Sl   Mar26 00:02:24 python /opt/enstore_cache/sbin/lm_director
enstore   7385  0.1  0.0 159464  7385    2 ?      Sl   Mar26 00:02:50 python /opt/enstore_cache/sbin/lm_director
enstore   7856  0.0  0.0 161368  7856    2 ?      Sl   Mar26 00:00:00 python /opt/enstore_cache/sbin/udp_proxy_server
```

```
lmd.udp_proxy_server
enstore   7856  0.0  0.0 161368  7888    2 ?        Sl   Mar26 00:00:00 python /opt/enstore_cache/sbin/udp_proxy_server
lmd.udp_proxy_server
enstore   7863  0.0  0.0 247380  7863    3 ?        Sl   Mar26 00:00:00 python /opt/enstore_cache/sbin/udp_proxy_server
lmd.udp_proxy_server
enstore   7863  0.0  0.0 247380  7900    3 ?        Sl   Mar26 00:00:12 python /opt/enstore_cache/sbin/udp_proxy_server
lmd.udp_proxy_server
enstore   7863  0.0  0.0 247380  8137    3 ?        Sl   Mar26 00:00:09 python /opt/enstore_cache/sbin/udp_proxy_server
lmd.udp_proxy_server
enstore   7879  0.0  0.0 151124  7879    1 ?        S    Mar26 00:00:20 python /opt/enstore_cache/sbin/udp_proxy_server
lmd.udp_proxy_server
enstore   7880  0.0  1.1 335680  7880    2 ?        Sl   Mar26 00:01:10 python /opt/enstore_cache/sbin/udp_proxy_server
lmd.udp_proxy_server
enstore   7880  0.0  1.1 335680  7898    2 ?        Sl   Mar26 00:01:40 python /opt/enstore_cache/sbin/udp_proxy_server
lmd.udp_proxy_server
enstore   7882  0.0  1.0 316088  7882    2 ?        Sl   Mar26 00:01:59 python /opt/enstore_cache/sbin/udp_proxy_server
lmd.udp_proxy_server
enstore   7882  0.0  1.0 316088  7899    2 ?        Sl   Mar26 00:00:08 python /opt/enstore_cache/sbin/udp_proxy_server
lmd.udp_proxy_server
root      1187  0.0  0.0 262212  1187    5 pts/4    Sl   10:54 00:00:00 python /opt/enstore_cache/sbin/migrator M3.migrator
root      1187  0.0  0.0 262212  1194    5 pts/4    Sl   10:54 00:00:00 python /opt/enstore_cache/sbin/migrator M3.migrator
root      1187  0.0  0.0 262212  1195    5 pts/4    Sl   10:54 00:00:00 python /opt/enstore_cache/sbin/migrator M3.migrator
root      1187  0.0  0.0 262212  1196    5 pts/4    Sl   10:54 00:00:00 python /opt/enstore_cache/sbin/migrator M3.migrator
root      1187  0.0  0.0 262212  1197    5 pts/4    Sl   10:54 00:00:00 python /opt/enstore_cache/sbin/migrator M3.migrator
root      1218  0.0  0.0 262224  1218    5 pts/4    Sl   10:54 00:00:00 python /opt/enstore_cache/sbin/migrator M5.migrator
root      1218  0.0  0.0 262224  1225    5 pts/4    Sl   10:54 00:00:00 python /opt/enstore_cache/sbin/migrator M5.migrator
root      1218  0.0  0.0 262224  1226    5 pts/4    Sl   10:54 00:00:00 python /opt/enstore_cache/sbin/migrator M5.migrator
root      1218  0.0  0.0 262224  1227    5 pts/4    Sl   10:54 00:00:00 python /opt/enstore_cache/sbin/migrator M5.migrator
root      1218  0.0  0.0 262224  1228    5 pts/4    Sl   10:54 00:00:00 python /opt/enstore_cache/sbin/migrator M5.migrator
root      1246  0.0  0.0 262076  1246    5 pts/4    Sl   10:54 00:00:00 python /opt/enstore_cache/sbin/migrator M1.migrator
root      1246  0.0  0.0 262076  2681    5 pts/4    Sl   11:04 00:00:00 python /opt/enstore_cache/sbin/migrator M1.migrator
root      1246  0.0  0.0 262076  2682    5 pts/4    Sl   11:04 00:00:00 python /opt/enstore_cache/sbin/migrator M1.migrator
root      1246  0.0  0.0 262076  2683    5 pts/4    Sl   11:04 00:00:00 python /opt/enstore_cache/sbin/migrator M1.migrator
root      1246  0.0  0.0 262076  2684    5 pts/4    Sl   11:04 00:00:00 python /opt/enstore_cache/sbin/migrator M1.migrator
```

4.3 Stopping enstore cache components

4.3.1.1 AMQP broker

AMQP broker can be stopped by the following command (under root account):

```
[root@pmig01]# /etc/init.d/qpid_broker stop
```

4.3.2 Enstore Cache Servers.

Enstore cache servers stop by “enstore stop ...” command as any other enstore server (under enstore account).

4.4 Monitoring enstore cache components

4.4.1.1 AMQP broker

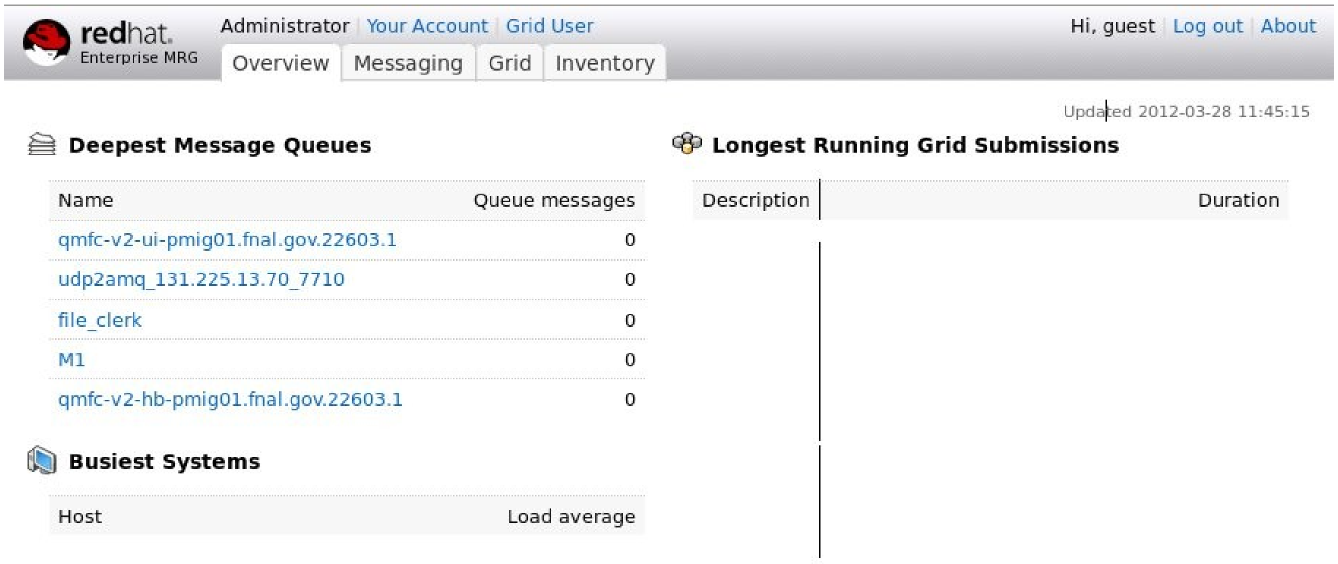
AMQP broker is monitored by qpid console on the host where the broker runs.

The console starts automatically, but can also be started by the following commands:

```
[root@pmig01]# source /opt/enstore_cache/cumin/setup-cumin
[root@pmig01]# cumin-database start
[root@pmig01]# cumin --console > /var/log/cumin/master.log 2>&1 &
```

Monitoring of qpid broker status and queues can be done over web interface. For this start firefox on the host where the broker runs and use the following URL:http://localhost:45672.

The page looks like:


redhat. Enterprise MRG
Administrator | Your Account | Grid User
Hi, guest | Log out | About
Overview | Messaging | Grid | Inventory

Updated 2012-03-28 11:45:15

Deepest Message Queues

Name	Queue messages
qmfc-v2-ui-pmig01.fnal.gov.22603.1	0
udp2amq_131.225.13.70_7710	0
file_clerk	0
M1	0
qmfc-v2-hb-pmig01.fnal.gov.22603.1	0

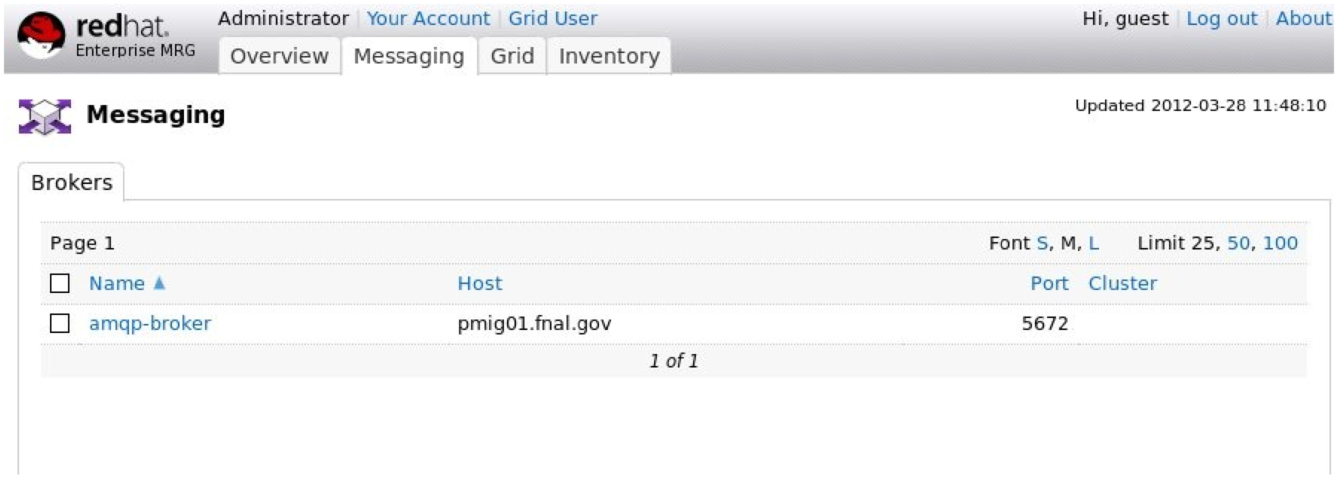
Busiest Systems

Host	Load average

Longest Running Grid Submissions

Description	Duration

Select tab “Messaging”:



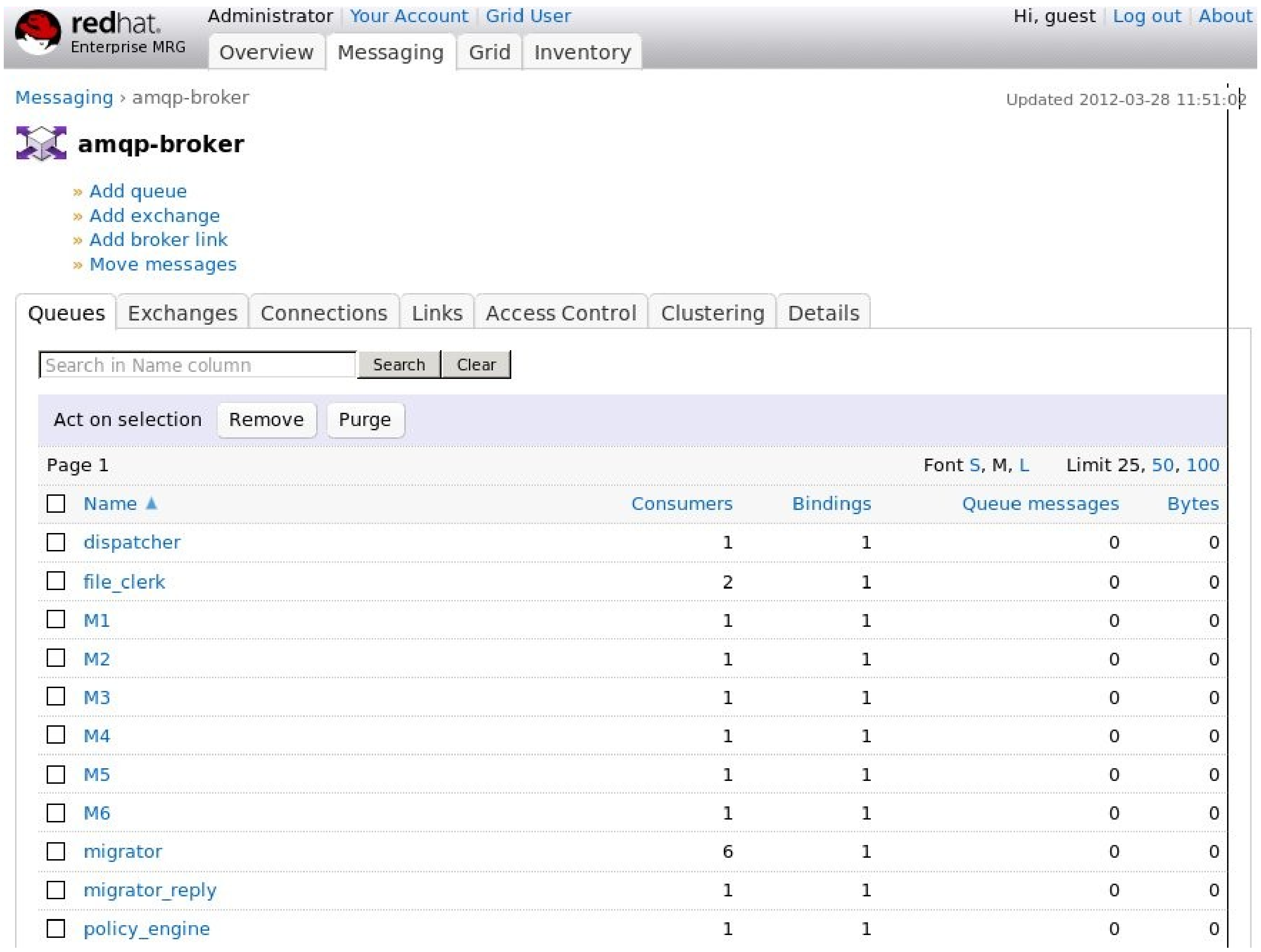
Updated 2012-03-28 11:48:10

Page 1 Font S, M, L Limit 25, 50, 100

	Name ▲	Host	Port	Cluster
☐	amqp-broker	pmig01.fnal.gov	5672	

1 of 1

Select “amqp-broker”



Updated 2012-03-28 11:51:02

Page 1 Font S, M, L Limit 25, 50, 100

	Name ▲	Consumers	Bindings	Queue messages	Bytes
☐	dispatcher	1	1	0	0
☐	file_clerk	2	1	0	0
☐	M1	1	1	0	0
☐	M2	1	1	0	0
☐	M3	1	1	0	0
☐	M4	1	1	0	0
☐	M5	1	1	0	0
☐	M6	1	1	0	0
☐	migrator	6	1	0	0
☐	migrator_reply	1	1	0	0
☐	policy_engine	1	1	0	0

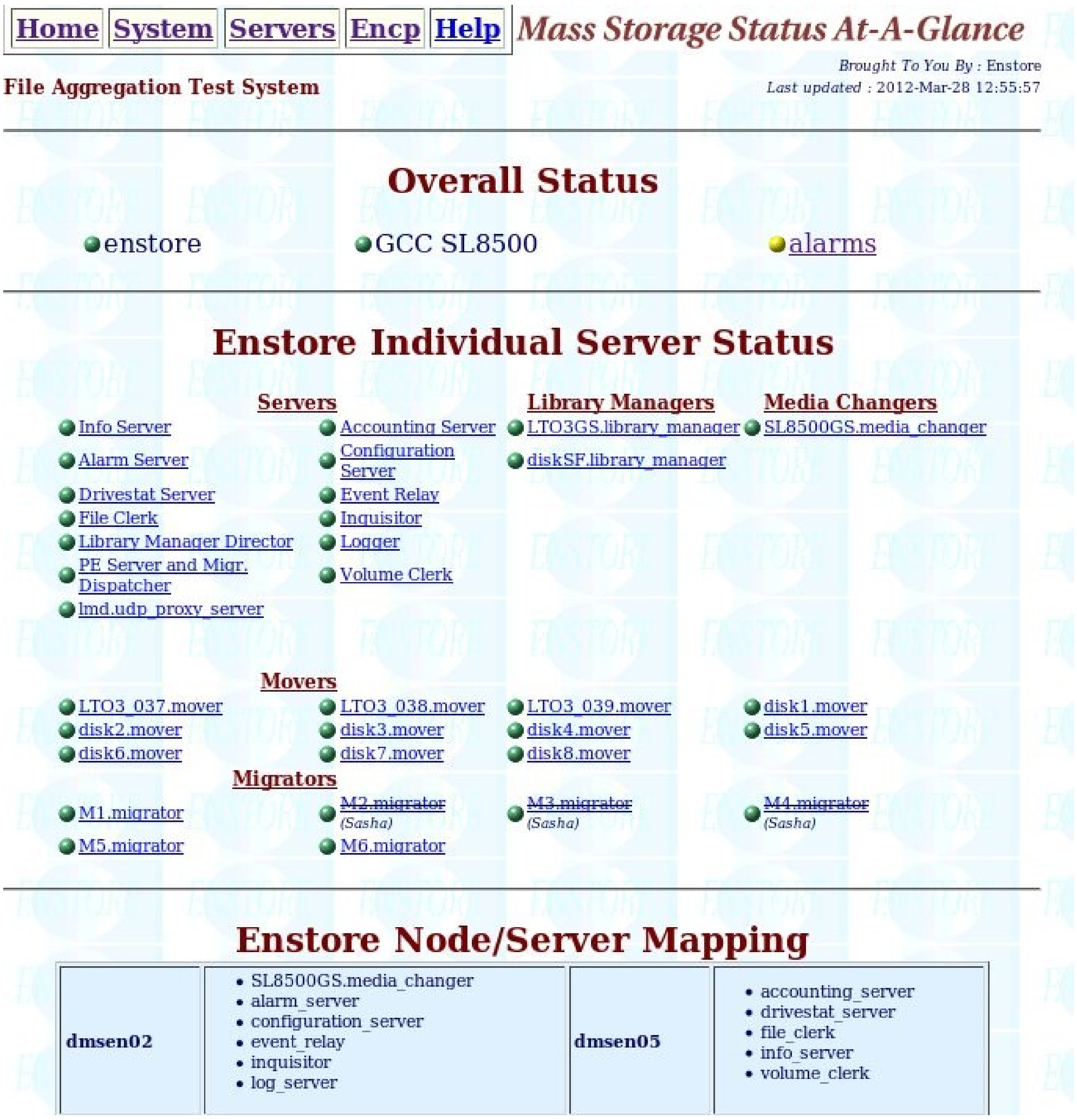
On this page you can see different queues and their status. The columns of interest are: “Queue Messages” and “Bytes”. “Queue messages” show how many messages are queued. “Bytes” show how many bytes are queued.

You can see all queues described in configuration plus individual queue for each migrator named by its name in configuration.

4.4.2 Enstore Cache Servers.

The state of enstore cache servers can be checked at enstore monitoring web pages.

Status at a glance page (fragment):



Home | System | Servers | Encp | Help

Mass Storage Status At-A-Glance

Brought To You By : Enstore

File Aggregation Test System

Last updated : 2012-Mar-28 12:55:57

Overall Status

● enstore ● GCC SL8500 ● alarms

Enstore Individual Server Status

Servers

● Info Server ● Accounting Server
● Alarm Server ● Configuration Server
● Drivestat Server ● Event Relay
● File Clerk ● Inquisitor
● Library Manager Director ● Logger
● PE Server and Migr. Dispatcher ● Volume Clerk
● lmd.udp_proxy_server

Library Managers

● LTO3GS.library_manager
● diskSF.library_manager

Media Changers

● SL8500GS.media_changer

Movers

● LTO3_037.mover ● LTO3_038.mover ● LTO3_039.mover ● disk1.mover
● disk2.mover ● disk3.mover ● disk4.mover ● disk5.mover
● disk6.mover ● disk7.mover ● disk8.mover

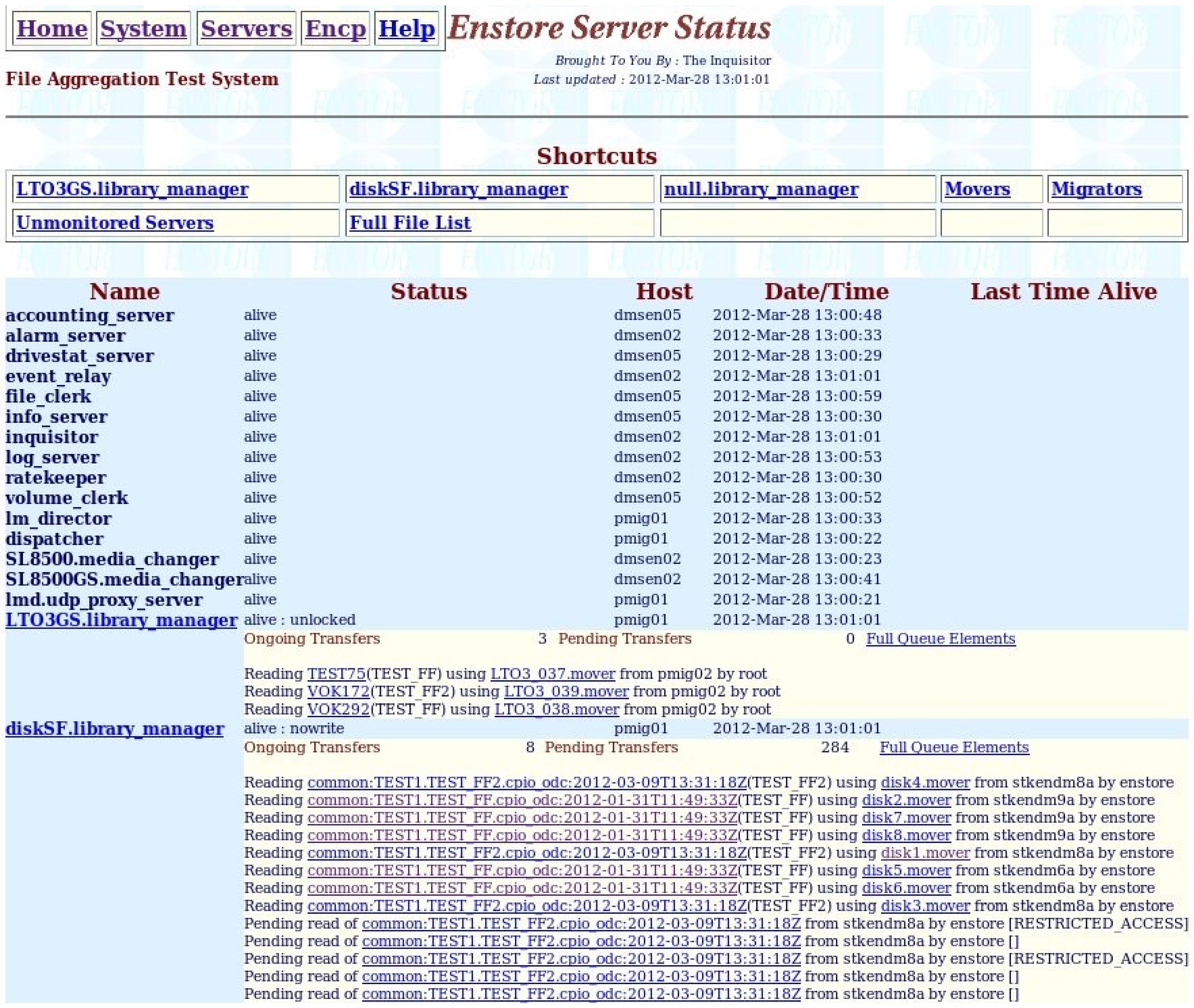
Migrators

● M1.migrator ● ~~M2.migrator~~ *(Sasha)* ● ~~M3.migrator~~ *(Sasha)* ● ~~M4.migrator~~ *(Sasha)*
● M5.migrator ● M6.migrator

Enstore Node/Server Mapping

dmsen02	• SL8500GS.media_changer • alarm_server • configuration_server • event_relay • inquisitor • log_server	**dmsen05**	• accounting_server • drivestat_server • file_clerk • info_server • volume_clerk

Enstore Servers page (fragment):



Home | System | Servers | Encp | Help

Enstore Server Status

Brought To You By : The Inquisitor
Last updated : 2012-Mar-28 13:01:01

File Aggregation Test System

Shortcuts

LTO3GS.library_manager	diskSF.library_manager	null.library_manager	Movers	Migrators
Unmonitored Servers	Full File List			

Name	Status	Host	Date/Time	Last Time Alive
accounting_server	alive	dmsen05	2012-Mar-28 13:00:48	
alarm_server	alive	dmsen02	2012-Mar-28 13:00:33	
drivestat_server	alive	dmsen05	2012-Mar-28 13:00:29	
event_relay	alive	dmsen02	2012-Mar-28 13:01:01	
file_clerk	alive	dmsen05	2012-Mar-28 13:00:59	
info_server	alive	dmsen05	2012-Mar-28 13:00:30	
inquisitor	alive	dmsen02	2012-Mar-28 13:01:01	
log_server	alive	dmsen02	2012-Mar-28 13:00:53	
ratekeeper	alive	dmsen02	2012-Mar-28 13:00:30	
volume_clerk	alive	dmsen05	2012-Mar-28 13:00:52	
lm_director	alive	pmig01	2012-Mar-28 13:00:33	
dispatcher	alive	pmig01	2012-Mar-28 13:00:22	
SL8500.media_changer	alive	dmsen02	2012-Mar-28 13:00:23	
SL8500GS.media_changer	alive	dmsen02	2012-Mar-28 13:00:41	
lmd.udp_proxy_server	alive	pmig01	2012-Mar-28 13:00:21	
LTO3GS.library_manager	alive : unlocked	pmig01	2012-Mar-28 13:01:01	

```
Ongoing Transfers      3  Pending Transfers      0  Full Queue Elements

Reading TEST75(TEST_FF) using LTO3_037.mover from pmig02 by root
Reading VOK172(TEST_FF2) using LTO3_039.mover from pmig02 by root
Reading VOK292(TEST_FF) using LTO3_038.mover from pmig02 by root
```

Name	Status	Host	Date/Time	Last Time Alive
diskSF.library_manager	alive : nowrite	pmig01	2012-Mar-28 13:01:01	

```
Ongoing Transfers      8  Pending Transfers      284  Full Queue Elements

Reading common:TEST1.TEST_FF2.cpio_odc:2012-03-09T13:31:18Z(TEST_FF2) using disk4.mover from stkendm8a by enstore
Reading common:TEST1.TEST_FF.cpio_odc:2012-01-31T11:49:33Z(TEST_FF) using disk2.mover from stkendm9a by enstore
Reading common:TEST1.TEST_FF.cpio_odc:2012-01-31T11:49:33Z(TEST_FF) using disk7.mover from stkendm9a by enstore
Reading common:TEST1.TEST_FF.cpio_odc:2012-01-31T11:49:33Z(TEST_FF) using disk8.mover from stkendm9a by enstore
Reading common:TEST1.TEST_FF2.cpio_odc:2012-03-09T13:31:18Z(TEST_FF2) using disk1.mover from stkendm8a by enstore
Reading common:TEST1.TEST_FF.cpio_odc:2012-01-31T11:49:33Z(TEST_FF) using disk5.mover from stkendm6a by enstore
Reading common:TEST1.TEST_FF.cpio_odc:2012-01-31T11:49:33Z(TEST_FF) using disk6.mover from stkendm6a by enstore
Reading common:TEST1.TEST_FF2.cpio_odc:2012-03-09T13:31:18Z(TEST_FF2) using disk3.mover from stkendm8a by enstore
Pending read of common:TEST1.TEST_FF2.cpio_odc:2012-03-09T13:31:18Z from stkendm8a by enstore [RESTRICTED_ACCESS]
Pending read of common:TEST1.TEST_FF2.cpio_odc:2012-03-09T13:31:18Z from stkendm8a by enstore []
Pending read of common:TEST1.TEST_FF2.cpio_odc:2012-03-09T13:31:18Z from stkendm8a by enstore [RESTRICTED_ACCESS]
Pending read of common:TEST1.TEST_FF2.cpio_odc:2012-03-09T13:31:18Z from stkendm8a by enstore []
Pending read of common:TEST1.TEST_FF2.cpio_odc:2012-03-09T13:31:18Z from stkendm8a by enstore []
```

Enstore alarms is raised if any of its servers is down, the e-mail is also sent in this case just as for any existing enstore servers.

4.5 Monitoring and controlling write requests.

All files written into cache get registered in files_in_transition table in enstore DB. When file is written to tape the corresponding entry is removed from files_in_transition table. File clerk periodically checks this table an generates warning alarms: “N files stuck in files_in_transition table”, where N is the total number of such files. The list of bfids in files_in_transition table is referred to in “Additional Information” column on alarms page. Note that this is just a warning and usually does not require any action. However if the number of such files continuously grows this may mean that some investigation is needed to find the reason. The possible actions may be:

Check SAAG and Servers web pages

Check statuses of migrators (enstore mig –status) to find out if they do any work.

If migrator is down start it (enstore start).

Check queues in dispatcher (enstore disp –get-q). If it does not have CACHE_MISSED lists in migration pool, replay events for files_in_transition table by the following command:

enstore file –replay

If packaging and migration to tapes does not resume (enstore mig –status <migrator> is one of the ways to check this) contact developers.

4.6 Emergency data migration to tape.

Some system operations, such as scheduled or unscheduled system shutdowns may require flushing all files not yet written to tape to get written even if criteria defined in the Policy Engine Server were not met. The special enstore command and the corresponding event will be designed for this.

Enstore command:

enstore disp –start-draining - this command instructs the policy engine to send all its current lists for migration to tapes.

4.7 Migration to new media.

In enstore there is a need of consistent migration of data form one tape media to another as the old media and drives get replaced with new technology. The natural way of providing such migration for packaged files is to stage them to cache with subsequent repackaging and archiving to a new tape. This approach does not require any changes in migration scripts.

5 Document Change log

v2	03/28/12	Sasha	Initial release
v3	04/03/12	Sasha	Changed per comments from John